Software Requirements Specification

For

Hostel Management System

Prepared by

Group Name: Code Monkeys

KALYAN KRISHNA	SE22UCSE302	se22ucse302@mahindrauniversity.edu.in
TEJAS	SE22UCSE270	se22ucse270@mahindrauniversity.edu.in
SRIROOP	SE22UCSE190	se22ucse190@mahindrauniversity.edu.in
MUKUND	SE22UCSE075	se22ucse075@mahindrauniversity.edu.in
MOHANA KRISHNA	SE22UCSE173	se22ucse173@mahindrauniversity.edu.in
VIKRANTH	SE22UCSE009	se22ucse009@mahindrauniversity.edu.in
CHANDRAKIRAN	SE22UCSE134	se22ucse134@mahindrauniversity.edu.in
VARUN	SE22UCSE069	se22ucse069@mahindrauniversity.edu.in

Instructor: Vijay Rao

Course: Software Engineering

Lab Section: *CSE*

Teaching Assistant: *Vijay Rao*

Date: 9-03-2025

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	**KALYAN KRISHNA**	Final Review and formatting improvements.	06/03/25

1 Introduction

1.1 Document Purpose

The Hostel Management System (HMS) is an online platform that allows freshers to select rooms, pay fees, and receive receipts, while wardens can manage hostel occupancy, view available rooms, and search by student ID or room number. This Software Requirements Specification (SRS) outlines the system's features, including room allocation, student registration, fee management, and complaint handling. It defines the system requirements, intended users, and technical considerations, serving as a guide for developers, testers, and stakeholders to ensure a well-structured and efficient digital solution.

1.2 Product Scope

The Hostel Management System is a web-based application designed to manage hostel accommodations and related administrative tasks efficiently. The system aims to streamline student room allocation, fee management, and communication between students and hostel administration. By automating these processes, the system reduces manual workload, minimizes errors, and improves the overall efficiency of hostel management.

1.3 Intended Audience and Document Overview

Developers: Responsible for designing, developing, and implementing the system based on the requirements specified in this document.
Clients (University/Hostel Administration): Evaluate the system's functionalities to ensure it meets their operational needs.
Professors/Advisors: Review the document for academic evaluation and provide feedback for improvements.
Testers: Develop and execute test cases to validate the system's functionality, performance, and reliability.
End Users (Students & Hostel Staff): Interact with the system for room booking, fee payments, and complaint management.

1.4 Definitions, Acronyms and Abbreviations

Admin: The hostel administrator who manages student accommodations.

API: Application Programming Interface.

DBMS: Database Management System used to store and manage data.

HMS: Hostel Management System.

LMS: Login Management System, responsible for user authentication.

SQL: Structured Query Language used for database operations.

UI: User Interface.

UX: User Experience, which focuses on design and usability.

Web App: Web-based application accessible via a browser.

User: A student or staff member using the system.

RBAC: Role-Based Access Control

1.5 Document Conventions

This document follows the IEEE formatting guidelines for software requirement specifications, using Arial font with a size of 11 or 12 throughout. The text is single-spaced with 1-inch margins on all sides for consistency and readability. Section headings (e.g., **1. INTRODUCTION**) and subsections (e.g., **1.1 DOCUMENT PURPOSE**) are in bold uppercase to maintain a structured hierarchy. Emphasis is applied using italicized text for comments or placeholders, **bold text** for section titles and important terms, and monospace font for code snippets, if applicable.

1.6 References and Acknowledgments

IEEE Software Engineering Standards for requirement documentation.

Various online resources and academic materials related to hostel management systems.

Acknowledgments: Special thanks to our project mentors and team members for their valuable contributions.

2 Overall Description

2.1 Product Overview

The Hostel Management System is a software product designed to streamline the management of hostel operations. It is a new system developed to replace traditional manual record-keeping methods with a digitized, that improves efficiency, reduces errors, and enhances the user experience for both hostel administrators and students.

The system operates as a standalone website but has the potential to be integrated with existing university ERP systems for seamless data exchange and enhanced institutional efficiency. It includes features such as user authentication, RBAC, and real-time notifications to keep users informed about hostel-related activities.

~~2.2~~ Product Functionality

The Hostel Management System provides a comprehensive set of features to streamline hostel operations, improve efficiency, and enhance the user experience for administrators, wardens, and residents. Below is a high-level summary of its major functions:

User Management

Role-based access for admins, wardens, and students
Secure user authentication and profile management
Room & Allocation Management

Room swapping requests and approvals
Vacancy tracking and status updates
Student Information Management

Maintain student records, including personal details and room assignments
Fee & Payment Management

Automated fee calculation and invoice generation
Online payment gateway integration

2.3 Design and Implementation Constraints

1. Hardware Constraints

The system must be deployable on standard web servers with at least:
Processor: Minimum Intel i3 (or equivalent)
RAM: Minimum 2GB (Recommended: 4GB for better performance)
Internet: Minimum 2Mbps (Recommended 10Mbps for smooth access).
Works on mobile phones, but a desktop/laptop is recommended for administrative tasks.

2. Software & Technology Constraints

Programming Languages: The backend must be developed in Node.js with express.js, and the frontend in React.js.
Database: The system must use MongoDB or MySQL for data storage.
Web Frameworks: Express.js will be used for backend API development.
Authentication: OAuth 2.0 and JWT-based authentication must be implemented.
Third-party Integrations: The system must integrate with payment gateways (PayPal, Stripe, or Razorpay) for hostel fee collection.

3. Interface Constraints

The system must provide APIs to integrate with existing university ERP systems.
The user interface must be accessible via web browsers with support for Chrome, Firefox, and Edge.

4.Security Constraints

The system must implement JWT authentication and hashed passwords to ensure secure user login and data protection. RBAC must be enforced to restrict access based on user roles, preventing unauthorized actions. Additionally, data encryption should be applied to sensitive information, including passwords and payment details, to enhance security.

2.4 Assumptions and Dependencies

The development and deployment of the Hostel Management System rely on several assumptions and dependencies that could impact the project's requirements and implementation. If these assumptions turn out to be incorrect or if dependencies change, it could lead to modifications in design, development time, or cost.

Major Assumptions
Internet Connectivity is Available

The system assumes that users (administrators, wardens, and students) have access to a stable internet connection for accessing the web-based platform.
Users Have Basic Technical Knowledge

Hostel administrators and wardens are assumed to have basic computer literacy to operate the system.
Minimal training will be required to use the system effectively.
Institution Will Provide Necessary Hardware & Infrastructure

The organization using the system is assumed to have servers or cloud hosting available to deploy the application.
End users will access the system through modern web browsers (Chrome, Firefox, Edge, etc.).
Third-Party Services Will Remain Available and Functional

The system depends on third-party APIs for certain functionalities:
Payment Processing (e.g., PayPal, Stripe, Razorpay)
Email & SMS Notifications
If any of these services become unavailable, alternatives will need to be integrated.

Technology Stack
Backend: Node.js with Express.js
Frontend: HTML, CSS, React.js
Database: MongoDB or MySQL
Authentication: OAuth 2.0, JWT
Hosting & Deployment

It may also be deployed on institutional servers if required.
Third-Party Integrations

Payment Gateways for hostel fee collection.
SMS/Email APIs for notifications and alerts.
ERP System Integration (if applicable) for student data synchronization.
User Access and Devices

The system is designed for web-based access, assuming users will interact through laptops, desktops, and mobile browsers.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

1. Admin Dashboard
Features:
Overview of hostel occupancy, finances, and complaints.
Menu-based navigation for room allotment, payments.

2. Student Portal
Features:
Apply for rooms and view assigned room details.
View and pay hostel fees online.
Receive notifications for announcements and deadlines.

3. Interaction Methods
Touch-friendly UI for mobile/tablet accessibility.
Dropdown menus and sidebar navigation for easy access.

3.1.2 Hardware Interfaces

The Hostel Management System (HMS) primarily interacts with standard computing hardware to facilitate hostel operations. Below is a list of the hardware interfaces the system interacts with:

1. Server & Hosting Infrastructure
Cloud Servers (AWS, Azure, DigitalOcean) or On-Premise Servers for system hosting.
Database Servers (MySQL, MongoDB) to store hostel records.

2. User Devices
Desktop & Laptop Computers (Windows, macOS, Linux) for administrators and wardens.
Smartphones & Tablets (Android, iOS) for students accessing the system via a responsive web interface.

3.1.3 Software Interfaces

The Hostel Management System (HMS) interacts with various software components to provide seamless functionality. Below are the key software interfaces:

1. Web Application (Primary Interface)
Platform: Web-based (built using React.js for frontend and Node.js (Express.js) for backend).
Access: Admins, wardens, and students interact with the system via a browser.
Communication: The frontend communicates with the backend via APIs.

2. Mobile Application (Future Integration)
The system is designed to integrate with a mobile app for students and wardens.
The app will use API requests to:
View notifications (fee due dates, announcements, etc.).
Submit complaints and track their status.
Check room availability and apply for a room change.
Make hostel fee payments through integrated payment gateways.

3. Database Management System
Supported Databases: MongoDB (NoSQL) or MySQL (Relational).
Functionality: Stores all hostel-related data, including student records, room allocations, complaints, and payments.

4. Third-Party APIs & Services
Payment Gateway APIs (e.g., PayPal, Stripe, Razorpay) – for online payments.

3.2 Functional Requirements

Below are the detailed functional requirements of the Hostel Management System (HMS). Each function describes a specific feature that the system must provide to meet the project's objectives.

F1: User Authentication and Authorization

F1.1 The system shall allow users to register and log in using email and password.
F1.2 The system shall provide role-based access control (RBAC) for different user types (admin, warden, student).
F1.3 The system shall support OAuth 2.0 and JWT authentication for secure login sessions.
F1.4 The system shall allow students to reset their passwords via email verification.

F2: Hostel Room Management
F2.1 The system shall allow administrators to add, update, and remove hostel rooms.
F2.2 The system shall allow students to apply for hostel rooms through the online portal.
F2.3 The system shall automatically assign rooms based on availability and predefined rules (e.g., gender-based allocation).
F2.4 The system shall allow students to request a room change, which wardens can approve or reject.
F3: Student Profile and Records Management
F3.1 The system shall maintain a database of all students, including personal details, room assignments, and payment history.
F4: Fee Management and Online Payments
F4.1 The system shall generate fee invoices for hostel accommodation and display them to students.
F4.2 The system shall integrate with payment gateways (PayPal, Stripe, Razorpay) for online transactions.
F4.3 The system shall allow students to pay their hostel fees online and receive an automated payment receipt.

3.3 Use Case Model

Use Case 1: Select and Book a Bed (with Concurrency Control)

- **Author**: Kalyan Krishna (Frontend developer)
- **Purpose**: Allows a student to select and book an available bed in Block A, ensuring that no other student can book the same bed simultaneously. The objective is to provide a reliable, RedBus-style bed selection process with concurrency protection.
- **Requirements Traceability**:
  - Student workflow: Check Room Availability, Book Room.
  - Functional Requirement: Room availability check and booking.
  - Non-Functional Requirement: Prevent double-booking through concurrency control.
- **Priority**: High – Core functionality; critical for system integrity upon deployment.
- **Preconditions**:
  - Student is logged into the hostel management system with a valid account.
  - Block A has at least one available bed across its 5 floors.
  - Bed layout data is loaded and synchronized with the backend.
- **Post Conditions**:
  - The selected bed (e.g., G01-A) is exclusively booked by one student.
  - The bed's status updates to "booked" (red) in the layout for all users.
  - Booking is recorded with the student's ID and timestamp.
- **Actors**:
  - Student (human): Initiates bed selection and booking.
  - System (hostel management software): Manages bed status, enforces concurrency, and records booking.
- **Extends**: None (base use case).
- **Flow of Events**:

1. **Basic Flow**:
   1. Student navigates to the bed booking page for Block A.
   2. System displays a dropdown with floor options: Ground, 1st, 2nd, 3rd, 4th.
   3. Student selects a floor (e.g., Ground Floor).
   4. System renders the Ground Floor layout: 14 rooms (G01-G14), 2 beds each, color-coded (green = available, red = booked), based on real-time backend data.
   5. Student identifies an available bed (e.g., G01-A, green).
   6. Student clicks G01-A.

7. System sends a request to the backend to **reserve** G01-A temporarily (e.g., lock it for 30 seconds or mark it as "pending").
8. Backend checks if G01-A is still available:
    - If yes, reserves it and returns "Reserved for you."
    - If no (already reserved/booked), returns "Bed unavailable."
9. If reserved successfully, system prompts: "Confirm booking for Bed G01-A? (Expires in 30 seconds)"
10. Student clicks "Confirm" within the time limit.
11. System finalizes the booking, updates G01-A to "booked" (turns red), releases the reservation, and saves the booking with the student's ID.
12. System broadcasts the updated status (red) to all connected clients (e.g., via WebSocket or polling).
13. System displays: "Bed G01-A booked successfully."

2. **Alternative Flow**:
    - **A1: Bed Already Reserved**:
        - At step 8, if G01-A is reserved/booked by another student, system displays: "Bed G01-A is currently unavailable. Try another bed."
        - Student selects another bed (e.g., G02-A) and resumes at step 6.
    - **A2: Reservation Times Out**:
        - At step 10, if student doesn't confirm within 30 seconds, system releases the reservation, G01-A remains green, and displays: "Reservation expired. Please try again."
        - Student restarts at step 6 if desired.
    - **A3: Cancel Booking**:
        - At step 10, student clicks "Cancel" instead of "Confirm."
        - System releases the reservation, G01-A stays green, process ends.
3. **Exceptions**:
    - **E1: Network Failure During Reservation**:
        - At step 7, if the reservation request fails, system displays: "Failed to reserve bed. Retry?"
        - Bed remains available until retry succeeds.
    - **E2: Backend Conflict**:
        - At step 11, if another process books G01-A despite reservation (e.g., system lag), system rolls back, shows: "Booking failed due to conflict," and G01-A stays booked by the other student.

- **Includes**:
    - UC-002 (View Bed Availability by Floor)
    - UC-003 (Switch Between Floors)
- **Notes/Issues**:
    - **Implementation**: Requires backend API endpoints:
        - /reserve-bed
        - /confirm-booking (finalize booking,).
        - Real-time updates via WebSocket or polling
    - **Concurrency Mechanism**: Options include:
        - **Pessimistic Locking**: Lock the bed in the database during reservation.
        - **Optimistic Locking**: Use a version/timestamp check to ensure no changes since reservation.
    - **UI**: Add a "Pending" state (e.g., yellow bed) for reserved beds.
    - **Testing**: Simulate 50+ concurrent users to validate.

Use Case 2: View Bed Availability by Floor

- **Author**: Sriroop (Frontend developer)

- **Purpose**: Enables a student to view the availability of beds on a specific floor in Block A, helping them decide which bed to book. The goal is to present a clear, RedBus-style grid layout.
- **Requirements Traceability**:
  - Student workflow: Check Room Availability.
  - Functional Requirement: Room availability check.
- **Priority:** High – Essential for informed bed selection; critical for deployment.
- **Preconditions:**
  - Student is logged into the system.
  - Block A's bed data (5 floors, 140 beds) is loaded and up-to-date.
- **Post Conditions:**
  - Student sees the bed availability layout for the selected floor (e.g., 1st Floor: 101-114).
- **Actors:**
  - Student (human): Requests to view availability.
  - System (software): Displays the bed layout.
- **Extends:** None (base use case).
- **Flow of Events:**

1. **Basic Flow:**
   1. Student accesses the bed booking page for Block A.
   2. System shows a dropdown: Ground, 1st, 2nd, 3rd, 4th.
   3. Student selects a floor (e.g., 1st Floor).
   4. System displays the 1st Floor layout:
      - Left Side: Rooms 101-107, beds A and B.
      - Right Side: Rooms 108-114, beds A and B.
      - Green = available, red = booked.
   5. Student reviews availability (e.g., 101-A green, 101-B red).
   6. Student decides to book (proceeds to UC-001) or switches floors.
2. **Alternative Flow:**
   - A1: View Multiple Floors:
     - After step 5, student selects another floor (e.g., 2nd Floor).
     - System repeats steps 4-5 for the new floor.
3. **Exceptions:**
   - E1: Data Load Failure: At step 4, if layout fails to load, system shows: "Error loading floor data. Retry?"

- **Includes: None.**
- **Notes/Issues:**
  - Static HTML data needs replacement with dynamic API fetch.
  - Ensure layout scales for 50+ simultaneous users.

4 Other Non-functional Requirements

4.1 Performance Requirements

The Hostel Management System must meet the following performance requirements to ensure efficiency, scalability, and a seamless user experience:

General System Performance
P1. The system should load the dashboard within 5 seconds after a user logs in.
P2. Any action (such as booking a room, generating receipts, or searching for students) should be processed and responded to within 5 seconds under normal server load.
P3. The system should support at least 100 concurrent users without significant performance degradation.
P4. Queries fetching student and room data should execute within 5 seconds for up to 100 records.
Authentication & Security
P5. The system should authenticate a user within 5 seconds after credentials are entered.
P6. Session expiration and logout should occur within 2 seconds once triggered.
Billing & Payments
P7. Payment processing (including confirmation) should not take more than 5 seconds.
Scalability & Reliability
P8. The system should maintain an uptime of 99.9%, ensuring availability for hostel administrators and students.
P9. Response times should not exceed 5 seconds even under peak load (e.g., during admission periods).
P10. Any system notification (such as room allocation, fee due alerts) should be delivered to users within 3 seconds of the triggering event.

4.2 Safety and Security Requirements

S1. All users must authenticate using multi-factor authentication (MFA) (e.g., password + OTP sent via email/SMS) before accessing sensitive features.
S2. Role-based access control (RBAC) must be implemented to restrict admin privileges only to authorized personnel (e.g., hostel warden, admin staff).
S3. Users must be automatically logged out after 10 minutes of inactivity to prevent unauthorized access from unattended devices.
S4. The system should log all login attempts and notify users of suspicious login activities (e.g., login from an unrecognized device or location).
S5. Mobile app users should be required to enable biometric authentication (fingerprint/face ID) for faster and more secure logins.
S6. A data backup policy must be implemented with daily backups stored in an off-site secure location to prevent data loss due to accidental deletion or cyberattacks.
S7. The system should have disaster recovery protocols in place to restore operations within 2 hours in case of critical failure or attack.

4.3 Software Quality Attributes

The Hostel Management System (HMS) must adhere to high-quality standards to ensure efficiency, reliability, and ease of use. The following quality attributes define key aspects of the system's performance and usability.

4.3.1 Reliability
The system should ensure continuous availability with minimal downtime.
Failover mechanisms should be in place to handle server failures and minimize disruptions.
Transactions should be designed to prevent data loss or corruption.
Automated monitoring tools should detect and log system failures while providing real-time alerts.
4.3.2 Adaptability & Scalability
The system should be flexible enough to accommodate future feature expansions without requiring major architectural changes.
The database and application should be capable of handling increased user loads without performance degradation.
Deployment should be supported in both on-premises and cloud-based environments to meet different infrastructure needs.
4.3.3 Maintainability
The system should follow a modular architecture, allowing independent updates to different functionalities.
API endpoints should be well-documented to facilitate integration with third-party applications.
Code should be structured according to industry best practices to simplify debugging and future upgrades.
4.3.4 Usability & User Experience
The user interface should be intuitive and easy to navigate, ensuring that users can complete common tasks efficiently.
The system should be accessible on various devices, including desktops, tablets, and mobile phones.
A feedback mechanism should be provided to allow users to report usability issues and suggest improvements.
4.3.5 Security & Robustness
The system should be able to handle high traffic loads without crashing or experiencing performance degradation.
Input validation should be enforced to prevent common security vulnerabilities such as SQL injection and cross-site scripting.
Data recovery mechanisms should be in place to restore critical information in case of system failure.

Appendix A – Data Dictionary

1. Constants

NAME	Description	Value/Example
MAX_ROOMS	Maximum number of rooms the system can manage	600 - 1200
MAX_STUDENTS	Maximum number of students the system can accommodate	1200 - 2400
SESSION_TIMEOUT	Time before automatic user logout due to inactivity	Configurable
PAYMENT_DEADLINE	Deadline for students to complete hostel payments	Defined per term
ROLE_ADMIN	Role identifier for hostel administrators	"Admin"
ROLE_STUDENT	Role identifier for student users	"Student"

2. State Variables

Variable Name	Description	Possible States
ROOM_STATUS	Indicates whether a room is occupied or vacant	"Vacant", "Occupied", "Reserved", "Under Maintenance"
PAYMENTS_STATUS	Tracks whether a student has paid hostel fees	"Paid", "Unpaid", "Overdue"
USER_AUTH_STATUS	Represents user login session	"Logged In", "Logged Out", "Session Expired"
MAINTANENCE_REQUEST	Tracks the status of maintenance requests	"Pending", "In Progress", "Resolved"
NOTIFICATION_STATUS	Status of system notifications	"Unread", "Read"
BOOKING_STATUS	Status of a student's room booking request	"Approved", "Rejected", "Pending"

3. Inputs

Input Name	Description	Related Operations
STUDENT_ID	Unique identifier for each student, Login	Room Allocation, Payments
ROOM_ID	Unique identifier for each room	Room Allocation, Availability Check
USERNAME	User's login credentials	Authentication
PAYMENT_AMOUNT	Amount entered during hostel fee payment	Payment Processing
MAINTAINENCE_REQUEST	Description of a reported issue	Maintenance Logging
SEARCH_QUERY	Query entered by admin or student for room search	Search Operation

4. Outputs

Output Name	Description	Related Operations
ROOM_ASSIGNMENT	Allocated room details for a student	Room Booking, Availability Check
PAYMENT_RECIEPT	Confirmation of completed hostel fee payment	Payment Processing
NOTIFICATION	System-generated messages for users	Alerts, Reminders, Room Status Updates
BOOKING_STATUS	Displays whether a booking request is approved or rejected	Room Booking
MAINTAINENCE_UPDATE	Updates on maintenance request status	Maintenance Tracking
REPORT_GENERATION	Generates reports on student occupancy, payments, etc.	Admin Operations

Appendix B - Group Log

1. Meeting Logs

Date	Time	Location/Platform	Attendees	Agenda	Key Discussions & Decisions
17/02/2025	10:35AM	In-Person	All Members	Initial Planning	Defined project scope, assigned roles
19/02/2025	3:00PM	In-Person	All Members	Requirements Gathering	Discussed functional & non-functional requirements
01/03/2025	7:00PM	Google-Meet	All Members	System Design	Decided on database schema, system architecture
03/03/2025	10:35AM	In-Person	All Members	Documentation Review	Finalized SRS draft, reviewed feedback

2. Group Activities & Contributions

Date	Task/Activity	Members Involved	Status	Notes
19/02/2025	Research on hostel requirements	All Members	Completed	Gathered data from hostel admin
28/02/2025	Functional requirements draft	All Members	Completed	Needs review from all members
1/03/2025	System design documentation	All Members	Completed	Defined system architecture
03/03/2025	Security & performance analysis	All Members	Completed	Identified key security requirements